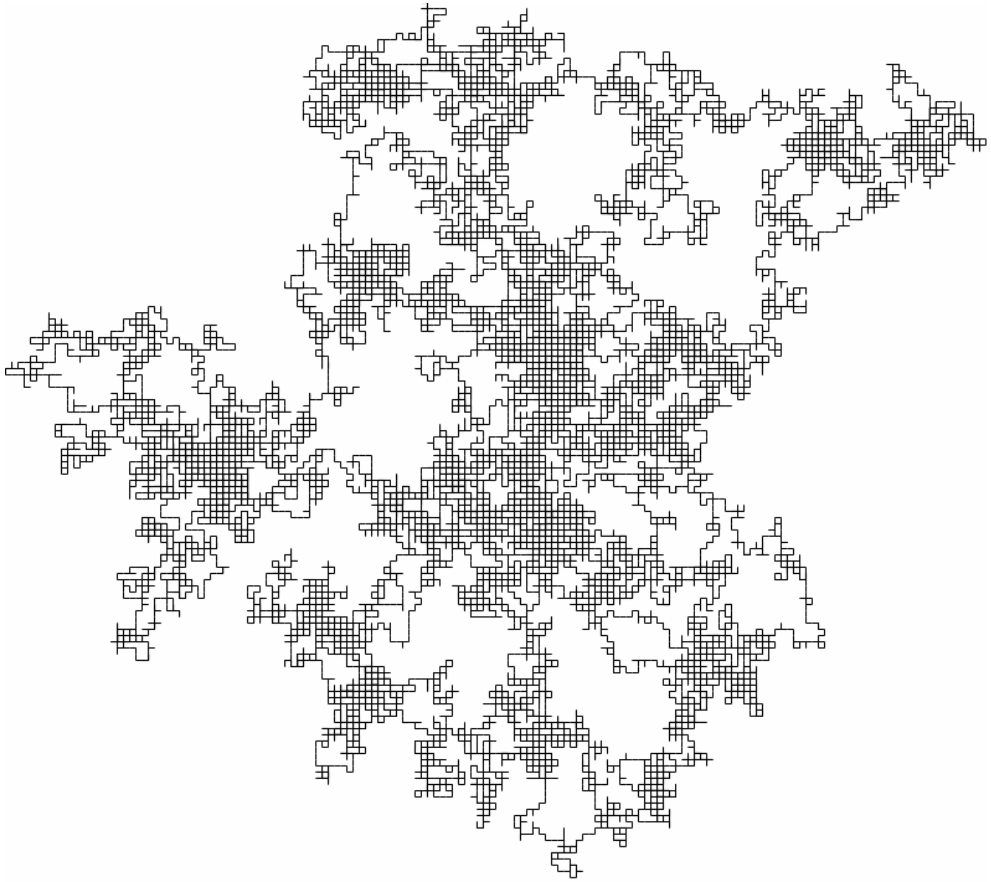
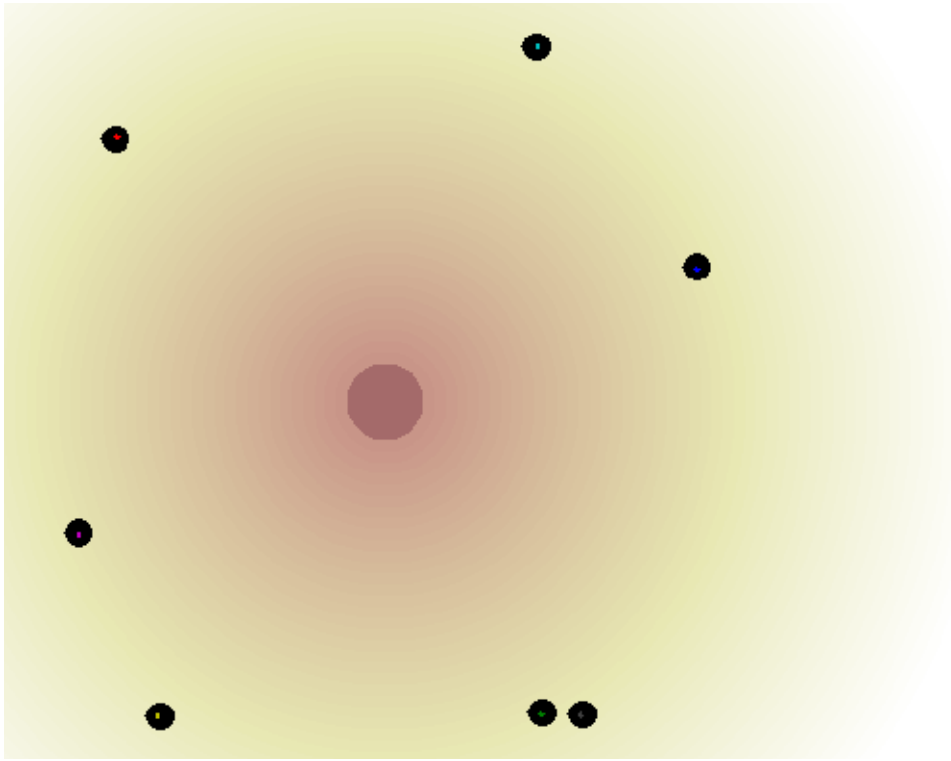
ICGE Module 2 Session 4

Scientific application of Python: Random Walks





Left:
https://commons.wikimedia.org/wiki/File:Random_walk_25000.gif

Right:
http://www.mit.edu/~kardar/teaching/projects/chemotaxis(AndreaSchmidt)/finding_food.htm

Many problems in physics, chemistry and biology essentially boil down to "random walks"

-4	-3	-2	-1	0	1	2	3	4

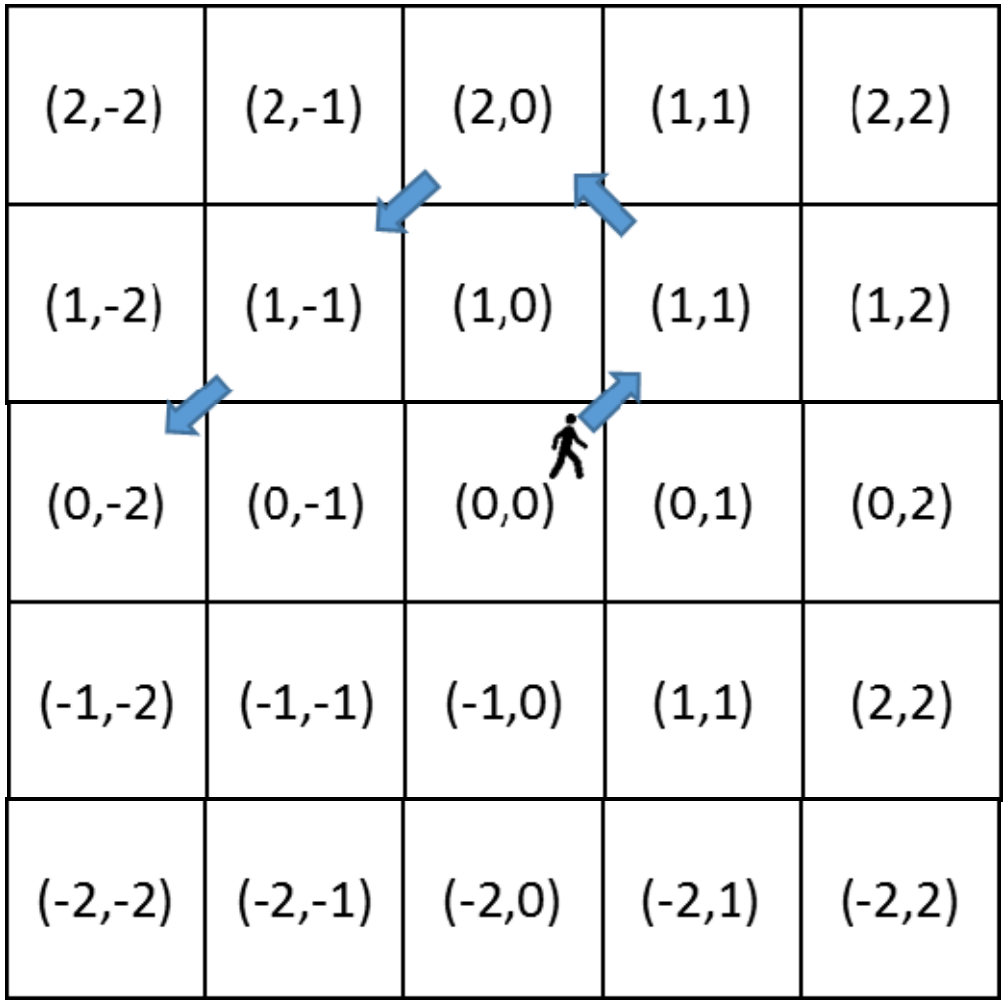
Simplest question to ask—does the walker ever get back home?

Python program for 1-D random walk

```
from __future__ import division
from random import choice
trials=1000
steps=1000
gothome=0
for i in range(trials):
    point=0
    for step in range(steps):
        point+=choice((-1,1))
        if point==0:
            gothome+=1
            break
print "Fraction that got home=%f" % (gothome/trials)
```

Save this as "**rwalk1d.py**" and run for different numbers of steps

The problem gets more interesting if the walker moves in 2 or more dimensions



Note that if we randomly change both x & y coordinates by -1 or +1, the walker moves diagonally like a checkers piece.

Python program for arbitrary-D random walk making diagonal moves at every step

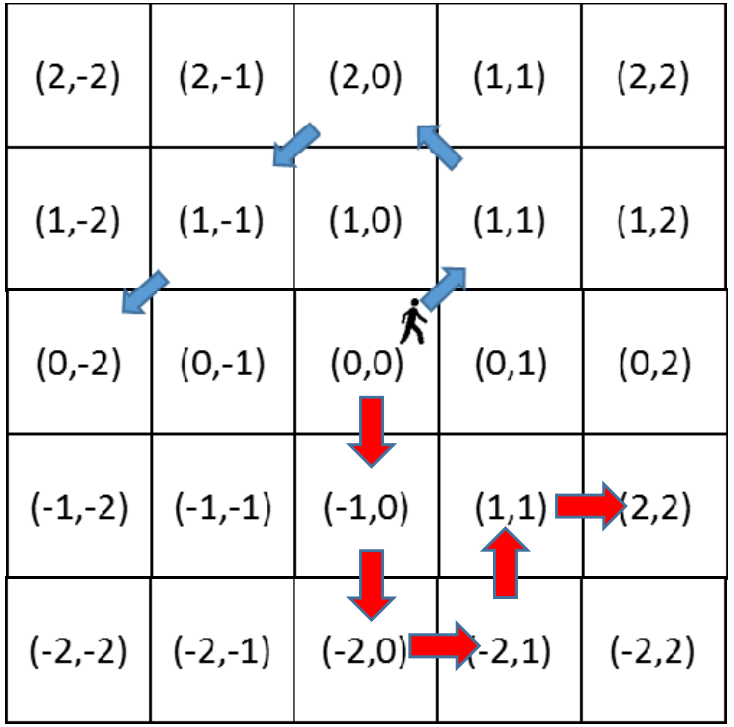
```
from __future__ import division
from random import choice
dim=3
trials=1000
steps=1000
gothome=0
for i in range(trials):
    point=[0]*dim
    for step in range(steps):
        for j in range(dim):
            point[j]+=choice((-1,1))
        if point.count(0)==dim:
            gothome+=1
            break
print "Fract that got home=%f in %d dims" % (gothome/trials,dim)
```

Save this as "**rwalknd.py**" and run for different dimensions

For what #'s of dimensions does the walker make it home?

Exact results for an infinite number of steps moving in only one dimension at a time

Dimensions	Prob(get home)
1	1.000
2	1.000
3	0.341
4	0.193
5	0.135
6	0.105
7	0.086
8	0.073



Research questions:

1. How well does **`rwalknd.py`** agree with the exact results?
2. Do your results match better if you modify the program to step in only one dimension at a time?

Exact results from http://mathworld.wolfram.com/PolyasRandomWalkConstants.html

Python program for arbitrary-D random moving in just one dimension each step

```
from __future__ import division
from random import choice
dim=3
trials=1000
steps=1000
gothome=0
for i in range(trials):
    point=[0]*dim
    for step in range(steps):
        movedim=choice(range(dim))
        point[movedim]+=choice((-1,1))
        if point.count(0)==dim:
            gothome+=1
            break
print "Fract that got home=%f in %d dims" % (gothome/trials,dim)
```

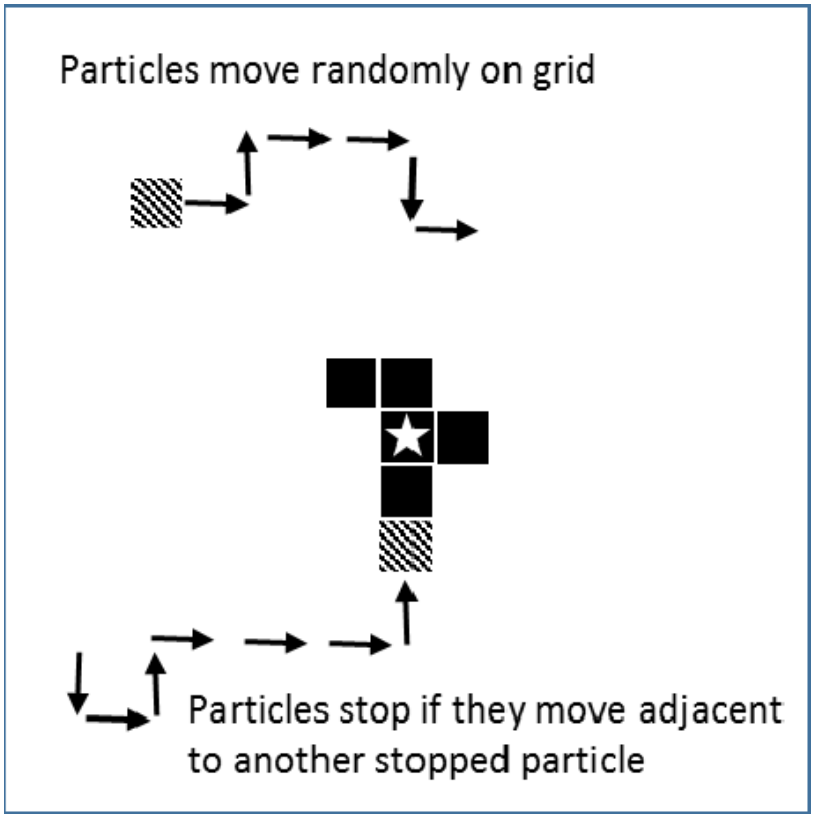
Save this as "**rwalknd2.py**" and run for different dimensions

Do these results agree better with pure theory?

Another important type of random walk simulation is diffusion-limited aggregation (DLA)

DLA "crystal" in copper sulfate solution

We can simulate DLA with a random 2-D walk where the particle stops if it hits an existing particle

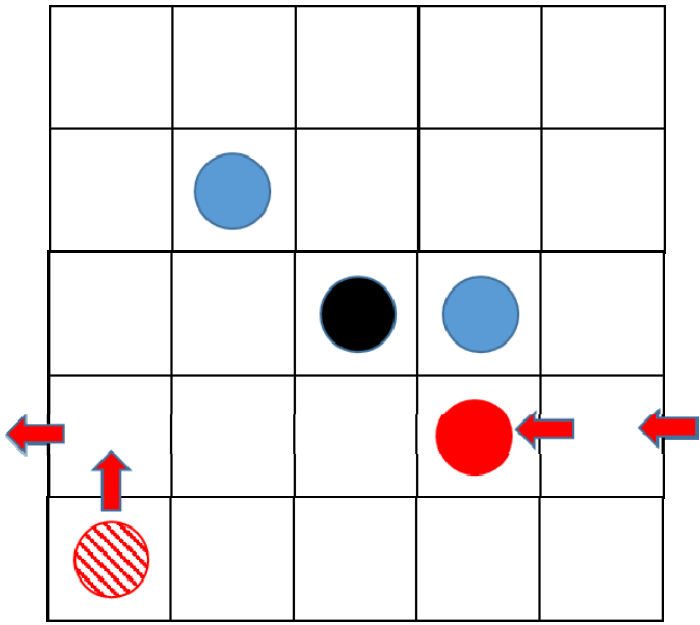


https://en.wikipedia.org/wiki/Diffusion-limited_aggregation

DLA Python program using very simple graphics based on the Python Image Library (PIL)

```
from random import choice
from drawgridlib import drawgrid, savegrid
npart=200  #Number of particles to aggregate
side=75  #Should be an odd number
steps = [(1,0),(-1,0),(0,1),(0,-1)]
grid=[[0 for x in range(side)] for y in range(side)]
grid[side/2][side/2]=1
for ipart in range(npart):
    # Start particle at origin
    x,y = 0,0
    # perform the random walk until particle aggregates
    while 1:
        grid[x][y]=0   #Remove particle from current spot
        # Randomly move particle
        sx,sy = choice(steps)
        x += sx
        y += sy
        # Enforce periodic boundaries
        if x < 0: x=side-1
        if y < 0: y=side-1
        if x==side: x=0
        if y==side: y=0
        grid[x][y]=1   #Put particle in new location
        # Stop if you are next to a particle
        if (grid[(x+1)%side][y]+grid[x][(y+1)%side]+
            grid[(x+side-1)%side][y]+grid[x][(y+side-1)%side])>0:
            break
drawgrid(grid,side)  #Displays image to screen
#savegrid(grid,side)  #Stores image in dla.jpg
```

Simulation grid



Output